# Bab 2

# Sekilas Model Relasional

Dalam bab ini akan dibahas sekilas mengenai model relasional. Tujuan utamanya hanya sebagai pengingat atau ringkasan saja. Pembahasan yang lebih mendalam dan menyeluruh dapat Anda peroleh di buku seperti buku berbahasa Inggris *Practical Issues in Database Management: A Reference for the Thinking Practitioner* karya Fabian Pascal atau *The Third Manifesto: Foundation for Object/Relational Databases* karya C. J. Date dan H. Darwen. Pemahaman akan model relasional tentu saja krusial dalam menguasai manajemen database, namun karena fokus buku ini adalah pada resep-resep SQL, diasumsikan pembaca sudah familiar dengan konsep-konsep model relasional.

# 2.1 Sejarah Model Relasional

Berkembangnya produk-produk database relasional seperti Oracle, Sybase, Ingres, DB/2 dan bahasa SQL sebetulnya adalah karena sebuah paper di tahun 1970. Paper yang kini menjadi bersejarah itu adalah karya E. F. Codd, karyawan departemen riset dan pengembangan IBM. Paper tersebut menjelaskan sebuah model untuk menyimpan data yang berbasiskan matematika, tepatnya teori set dan predicate logic.

Sebelum berkembangnya model relasional, dua model data yang paling dominan saat itu adalah model hirarkis dan model network. *Model hierarkis* menyimpan data dalam bentuk pohon, sama seperti yang dilakukan filesystem atau XML saat ini (jadi bisa dibilang XML itu bukan barang baru sama sekali, sebab sejak zaman pra-1970 pun sudah dipakai orang). Model ini amat cocok menyimpan jenis data tertentu yang memang berbentuk hierarkis seperti struktur organisasi perusahaan atau katalog-katalog. Namun model ini sebetulnya kaku, karena tidak cocok untuk menyimpan data lain yang tidak berbentuk seperti pohon.

Sementara model network menyimpan data dalam bentuk node-node yang dihubungkan satu sama lain; dengan kata lain, dalam bentuk graph. Model ini memang fleksibel karena amat generik, dapat menyimpan data dalam bentuk apa saja. Seperti kita ketahui, pohon adalah salah satu jenis graph, tapi sebuah graph dapat berbentuk macam-macam. Kekurangannya, model ini sulit dimengerti. Sebuah data yang kompleks dapat amat “njlimet” hubungan antarnodenya.

Kalau kita perhatikan, kalau kita menyimpan data di dalam struktur data di bahasa pemrograman (di memori), atau di dalam database objek, bentuknya adalah network. Tiap nilai skalar, atau tiap elemen hash atau array, merupakan node. Sementara tiap node dihubungkan dengan pointer atau referensi.

Di antara kedua model di atas, yang satu amat spesifik dan kaku sementara yang lain fleksibel tapi sulit dimengerti. Model relasional hadir sebagai jembatan atau alternatif yang lebih baik. Modelnya sederhana namun tetap fleksibel. Dan lagi memiliki landasan matematika yang jelas.

Saking sederhana dan fleksibelnya, berbagai pihak amat tertarik untuk mengembangkan produk database generasi baru berbasiskan model relasional. IBM sendiri, yang telah memiliki produk database berbasiskan model hierarkis/network, bermaksud mengembangkan produk database komersial yang berbasiskan model relasional ciptaan Codd ini, yang nantinya akan menggantikan produk lama. Namun seperti kita ketahui, sejarah mencatat bahwa IBM kalah cepat dari Oracle.

Oracle menjadi database relasional komersil pertama yang diluncurkan, sukses di pasaran, dan masih menjadi market leader sampai sekarang. Namun kehadiran Oracle segera disusul oleh vendor-vendor lain dan termasuk juga kini oleh komunitas open source. Hingga sekarang ada ratusan hingga ribuan produk database relasional dalam berbagai bentuk dan kemampuan.

## 2.2 Ringkasan Model Relasional

Seperti telah dijelaskan sebelumnya, model relasional sederhana-sederhana saja. Semua data disimpan dalam relasi. Sebuah relasi adalah predikat atau pernyataan kebenaran. Bagi para pembaca yang pernah bermain-main dengan bahasa AI Prolog tentu familiar dengan istilah predikat ini. Berikut ini contoh sebuah relasi:

```
Orang: {Nama, Kelamin}
```

Setelah itu kita dapat membuat tupel-tupel untuk relasi ini. Contoh:

```
(Amir, pria)
(Wati, wanita)
```

Tupel inilah yang merupakan butir-butir pernyataan atau data itu sendiri. Pada contoh di atas kita menaruh dua buah data/informasi, yaitu bahwa "Amir adalah seorang pria" dan "Wati adalah seorang wanita." Jadi semua data di dalam model relasional disimpan dalam bentuk tupel (di dalam relasi tertentu).

Relasi pada contoh di atas memiliki dua buah atribut, nama dan jenis kelamin. Contoh relasi lain di bawah ini:

```
Karyawan: {Nama, Kelamin, Gaji}
```

memiliki tiga buah atribut. Contoh tupel untuk relasi ini:

```
(Amir, pria, 1500000)
(Wati, wanita, 1000000)
(Burhan, pria, 3000000)
```

Sebuah atribut dapat memiliki nilai tertentu. Jenis nilai atribut disebut domain. Misalnya nama-nama karyawan disimpan dalam domain NamaKaryawan (atau string). Demikian pula domain-domain JenisKelamin dan angka.

Dalam istilah di teori set, dikatakan bahwa sebuah relasi adalah sebuah set berisi

tupel. Urutan tupel tidak penting dalam sebuah relasi. Tapi tentu saja urutan nilai atribut dalam sebuah tupel penting.

Dalam kenyataannya di SQL, sebuah relasi adalah sebuah multiset (“bag”), karena dapat mengandung tupel yang sama beberapa kali, misalnya:

```
(Amir, pria, 1500000)
(Amir, pria, 1500000)
```

Dan sebegitu sajalah deskripsi model relasional dalam menyimpan data: semua data disimpan dalam tupel di dalam relasi dan sebuah tupel adalah sekumpulan nilai atribut. Jadi, kalau dilihat, sebuah relasi berbentuk seperti tabel 2 dimensi. Tupel membentuk kolom, sementara relasi terdiri dari baris-baris tupel.

Sebetulnya dalam menaruh data ke dalam tupel dan relasi-relasi, Codd memberikan 13 buah aturan, jadi tidak sembarang saja. Misalnya, ada aturan yang menyebutkan bahwa sebuah tupel dalam sebuah relasi harus bisa diidentifikasi secara unik. Sebuah tabel SQL yang tidak memiliki primary key melanggar aturan ini karena mengizinkan lebih dari satu tupel yang sama persis untuk ada dalam relasi:

```
(Amir, pria, 1500000)
(Amir, pria, 1500000)
```

Pada contoh di atas, bagaimana cara merujuk pada data yang pertama atau data yang kedua? Karena tupelnya sama persis, maka ini tidak bisa dilakukan. Biasanya dalam SQL kita membuat atribut tambahan yaitu nomor unik dan menjadikannya sebagai primary key:

```
(1, Amir, pria, 1500000)
(2, Amir, pria, 1500000)
```

Sehingga antara data yang pertama dan kedua dapat dibedakan melalui nomor uniknya.

Selain ketiga belas aturan dalam menyimpan data di dalam tupel dan relasi, Codd juga memberi panduan bagaimana cara menyimpan data agar integritas data terlindungi. Panduan ini sering disebut sebagai “bentuk-bentuk normal” (normal forms). Ada beberapa tingkatan dan jenis bentuk normal, mulai dari 0NF (zeroth normal form), 1NF (first normal form), 2NF (second normal form), 3NF, 4NF, 5NF, DKNF (domain-key normal form), BCNF (Boyce-Codd Normal Form).

Pada intinya, dengan mengikuti bentuk-bentuk normal ini (dengan kata lain, dengan menjadikan relasi kita *normalized*) kita menghindari duplikasi data atau ketergantungan yang tak perlu antara sebuah data dan data lain. Sehingga memaksimumkan kemudahan pengubahan (tak perlu mengubah data yang sama di dua atau lebih tempat yang berbeda) dan sebetulnya meningkatkan performa.

## 2.3 Contoh Model Relasional

Kekuatan model relasional adalah dalam menghubungkan domain-domain yang sama yang ada pada relasi yang berbeda melalui operasi aljabar dan kalkulus relasional. Setiap relasi biasanya hanya menyimpan tupel-tupel berisi informasi yang secukupnya/seperlunya saja, jadi untuk meminta informasi tambahan yang berkaitan, kita dapat merujuk pada relasi lain. Sehingga duplikasi harus menyimpan informasi yang sama di dua buah relasi terhindari.

Kekuatan lain model relasional adalah dalam mengatur ketergantungan (dependency) antara sebuah atribut dengan atribut lain melalui fasilitas yang dinamakan referential integrity.

Mari melihat sebuah contoh:

```
Pelanggan: {KodePelanggan, Nama, Alamat, Telepon}
PesananPembelian: {NomorPesanan, KodePelanggan, Tanggal,
HargaTotal}
DetilPembelian: {NomorPesanan, NomorUrut, KodeBarang, Jumlah,
HargaItem}
Barang: {KodeBarang, NamaBarang, HargaBarang}
```

Di contoh di atas, kita mendefinisikan empat buah relasi. Pelanggan menyimpan daftar pelanggan. Pembelian menyimpan pesanan tiap pelanggan. DetilPembelian mencatat tiap jenis barang dalam sebuah pesanan. Dan Barang menyimpan daftar barang.

Perhatikan bahwa penyimpanan data seperti ini ternormalisasi. Kita tidak menyimpan nama atau alamat pelanggan secara dobel di PesananPembelian, melainkan kita hanya mencatat KodePelanggan saja. Manfaatnya? Manakala pelanggan memperbarui alamat atau nomor telepon atau data lainnya, kita tidak perlu memperbaruinya di banyak tempat. Bandingkan dengan hanya menyimpan kode pelanggan di PesananPembelian. Kalau ada perubahan data pelanggan, relasi PesananPembelian tidak perlu diutak-atik.

Data yang dobel-dobel merupakan salah satu biang masalah utama dalam manajemen data: kesulitan/kerepotan pengubahan data, ketidakkonsistenan data yang satu dengan yang lain, pemborosan tempat, dan sebagainya. Oleh Codd, manakala kita harus mengubah data yang sama di beberapa tempat sekaligus, hal tersebut dinamakan *update anomaly*. Jadi salah satu tujuan normalisasi adalah untuk menghindari update anomaly.

Kalau kita perhatikan, beberapa domain lain selain KodePelanggan juga disebutkan lebih dari satu kali di relasi yang berbeda: NomorPesanan dan KodeBarang. Ini juga untuk menghindari duplikasi data yang tak perlu.

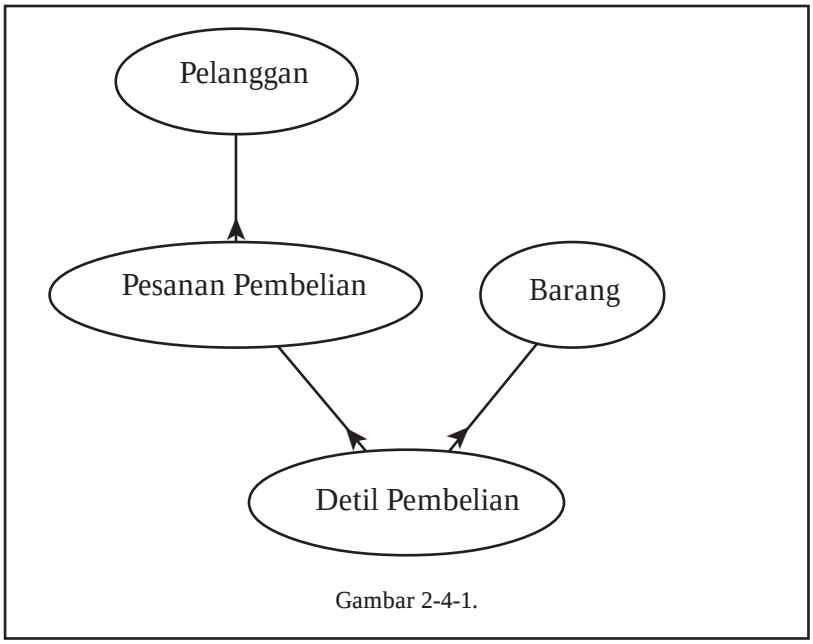
KodePelanggan data masternya berada di relasi Pelanggan, sementara KodePelanggan di relasi PesananPembelian bersifat tergantung (dependent) pada relasi Pelanggan. KodePelanggan di relasi PesananPembelian haruslah kode yang terdapat di relasi Pelanggan, tidak bisa sembarang kode. Hal ini dapat dijaga (di-*enforce*) di database relasional menggunakan *referential integrity*. Sehingga kita dapat yakin bahwa KodePelanggan di relasi PesananPembelian selalu valid.

Demikian pula halnya dengan KodeBarang di relasi Barang dan relasi DetilPembelian, dan NomorPesanan di relasi PesananPembelian dan DetilPembelian. Keduanya dijaga dengan referential integrity.

# 2.4 Kardinalitas Hubungan

Seperti telah kita lihat sebelumnya, antarrelasi dalam data relasional terhubung satu sama lain. Hubungan ini dapat berupa hubungan 1:1 (one-to-one), 1:M (one-to-many), atau M:M (many-to-many). Jenis hubungan ini lazim disebut sebagai kardinalitas hubungan.

Pada contoh sebelumnya, antara Pelanggan dan PesananPembelian terjadi hubungan dengan kardinalitas 1:M. Artinya, seorang pelanggan dapat memiliki lebih dari satu pesanan pembelian, tapi satu pesanan pembelian hanya terasosiasi dengan satu pelanggan saja. Demikian pula antara PesananPembelian dan DetilPembelian terdapat hubungan 1:M (sebuah pesanan terdiri dari beberapa baris pesanan, tapi sebuah baris pesanan hanya terasosiasi dengan satu buah pesanan). Contoh hubungan 1:1 misalnya antara relasi Pelanggan dan DataKartuMember. Sebuah pelanggan hanya boleh memiliki satu kartu member, demikian pula sebaliknya sebuah kartu member hanya untuk satu anggota. Contoh hubungan M:M misalnya antara relasi Anggota dan Grup, di mana sebuah anggota dapat terdaftar di beberapa grup dan sebuah grup dapat memiliki banyak anggota.




Gambar 2-4-1.

Perhatikan Gambar 2-4-1. Hubungan yang bersifat M (many) biasanya digambarkan dengan kaki burung (crow) pada sisi relasi yang bersangkutan.



## 2.4 Kardinalitas Hubungan